MPI Requirements of the Network Layer

Presented to the OpenFabrics libfabric Working Group
January 28, 2014

Community feedback assembled
by Jeff Squyres, Cisco Systems
Presented by
Nathan Hjelm, Los Alamos National Laboratory

Many thanks to the contributors (in no particular order)

- ETZ Zurich
  - Torsten Hoefler
- Sandia National Labs
  - Ron Brightwell
  - Brian Barrett
  - Ryan Grant
- IBM
  - Chulho Kim
  - Carl Obert
  - Michael Blocksome
  - Perry Schmidt
- Cisco Systems
  - Jeff Squyres
  - Dave Goodell
  - Reese Faucette
  - Cesare Cantu
  - Upinder Malhi
- Oak Ridge National Labs
  - Scott Atchley
  - Pavel Shamis
- Argonne National Labs
  - Jeff Hammond

Many thanks to the contributors (in no particular order)

- Intel
  - Sayantan Sur
  - Charles Archer
- Cray
  - Krishna Kandalla
- Mellanox
  - Devendar Bureddy
- SGI
  - Michael Raymond
- AMD
  - Brad Benton
- Microsoft
  - Fab Tillier
- U. Edinburgh / EPCC
  - Dan Holmes
- U. Alabama Birmingham
  - Tony Skjellum
  - Amin Hassani
  - Shane Farmer

Quick MPI overview

- High-level abstraction API
  - No concept of a connection
- All communication:
  - Is reliable
  - Has some ordering rules
  - Is comprised of typed messages
- Peer address is (communicator, integer) tuple
  - I.e., virtualized
  - Specifies a ***process***, not a ***server / network endpoint***

Quick MPI overview

- Communication modes
  - Blocking and non-blocking (polled completion)
  - Point-to-point: two-sided and one-sided
  - Collective operations: broadcast, scatter, reduce, …etc.
  - …and others, but those are the big ones
- Async. progression is required/strongly desired
- Message buffers are provided by the application
  - They are not “special” (e.g., registered)

Quick MPI overview

- MPI specification
  - Governed by the MPI Forum standards body
  - Currently at MPI-3.0
- MPI implementations
  - Software + hardware implementation of the spec
  - Some are open source, some are closed source
  - Generally don't care about interoperability (e.g., wire protocols)

MPI is a large community

- Community feedback represents union of:
  - Different viewpoints
  - Different MPI implementations
  - Different hardware perspectives
- …and not all agree with each other
- For example…

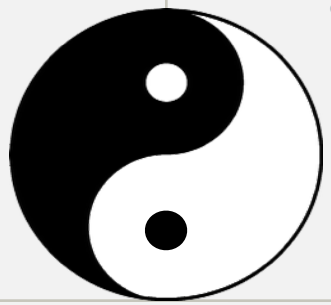
Different MPI camps

Those who want high level interfaces

- Do not want to see memory registration
- Want tag matching
  - E.g., PSM
  - Trust the network layer to do everything well under the covers

Those who want low level interfaces

- Want to have good memory registration infrastructure
- Want direct access to hardware capabilities
  - Want to fully implement MPI interfaces themselves
  - Or, the MPI implementers are the kernel / firmware /hardware developers



Be careful what you ask for…

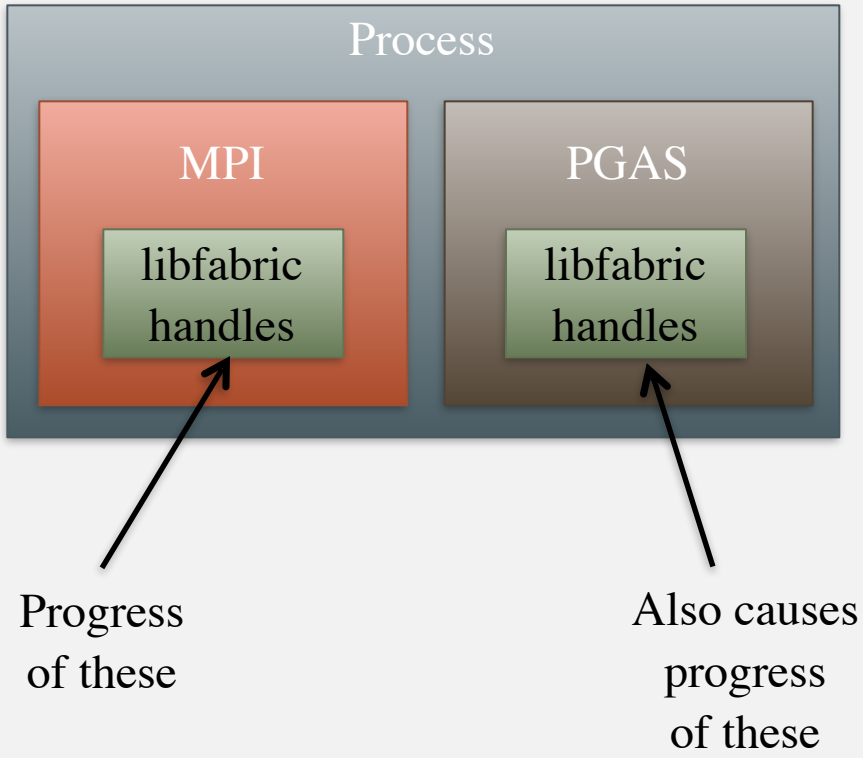
- …because you just got it
- Members of the MPI Forum would like to be involved in the libfabric design on an ongoing basis
- Can we get an MPI libfabric listserv?

Basic things MPI needs

- Messages (not streams)
- Efficient API
  - Allow for low latency / high bandwidth
  - Low number of instructions in the critical path
  - Enable “zero copy”
- Separation of local action initiation and completion
- One-sided (including atomics and shared locks) and two-sided semantics
- No requirement for communication buffer alignment (!!!)

Basic things MPI needs

- Asynchronous progress independent of API calls
  - Including asynchronous progress from multiple consumers (e.g., MPI and PGAS in the same process)
  - Preferably via dedicated hardware



Basic things MPI needs

- Scalable communications with millions of peers
  - With both one-sided and two-sided semantics
  - Think of MPI as a fully-connected model
    (even though it usually isn't implemented that way)
  - Today, runs with 3 million MPI ***processes*** in a job

Things MPI likes in verbs

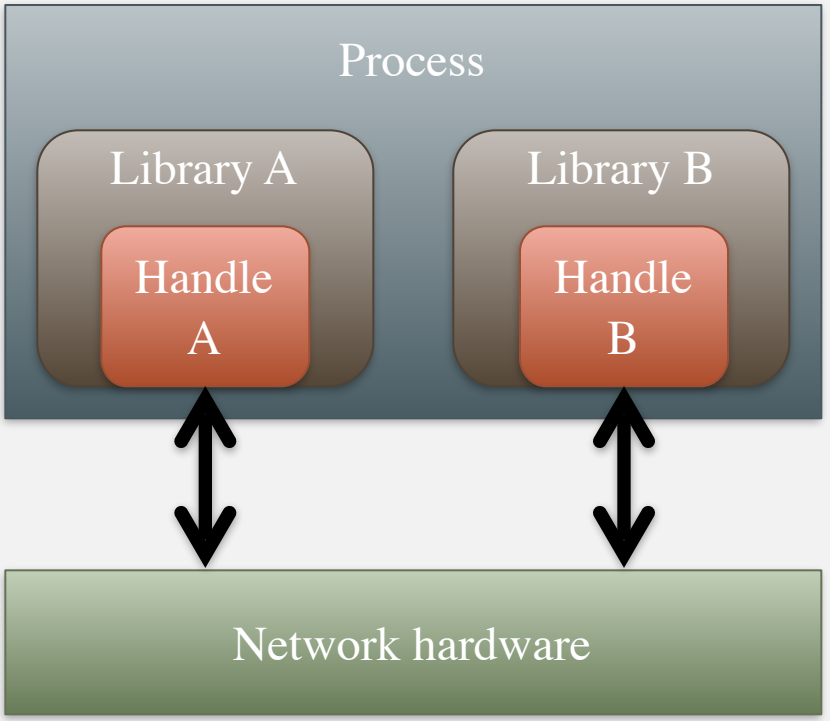
- (all the basic needs from previous slide)
- Different modes of communication
  - Reliable vs. unreliable
  - Scalable connectionless communications (i.e., UD)
- Specify peer read/write address (i.e., RDMA)
- RDMA write with immediate (*)
  - *…but we want more (more on this later)*

Things MPI likes in verbs

- Ability to re-use (short/inline) buffers immediately
- Polling and OS-native/fd-based blocking QP modes
- Discover devices, ports, and their capabilities (*)
  - *…but let's not tie this to a specific hardware model*
- Scatter / gather lists for sends
- Atomic operations (*)
  - *…but we want more (more on this later)*

Things MPI likes in verbs

- Can have multiple consumers in a single process
  - API handles are independent of each other



Things MPI likes in verbs

- Verbs does not:
  - Require collective initialization across multiple processes
  - Require peers to have the same process image
  - Restrict completion order vs. delivery order
  - Restrict source/target address region (stack, data, heap)
  - Require a specific wire protocol (*)
    - *…but it does impose limitations, e.g., 40-byte GRH UD header*

Things MPI likes in verbs

- Ability to connect to “unrelated” peers
- Cannot access peer (memory) without permission
- Ability to block while waiting for completion
  - *...assumedly without consuming host CPU cycles*
- Cleans up everything upon process termination
  - E.g., kernel and hardware resources are released

Other things MPI wants (described as verbs improvements)

- MTU is an int (not an enum)
- Specify timeouts to connection requests
  - *…or have a CM that completes connections asynchronously*
- All operations need to be non-blocking, including:
  - Address handle creation
  - Communication setup / teardown
  - Memory registration / deregistration

Other things MPI wants (described as verbs improvements)

- Specify buffer/length as function parameters
  - Specified as struct requires extra memory accesses
  - *…more on this later*
- Ability to query how many credits currently available in a QP
  - To support actions that consume more than one credit
- Remove concept of "queue pair"
  - Have standalone send channels and receive channels

Other things MPI wants (described as verbs improvements)

- Completion at target for an RDMA write
- Have ability to query if loopback communication is supported
- Clearly delineate what functionality *must* be supported vs. what is optional
  - Example: MPI provides (almost) the same functionality everywhere, regardless of hardware / platform
  - Verbs functionality is wildly different for each provider

Other things MPI wants (described as verbs improvements)

- Better ability to determine causes of errors
- In verbs:
  - Different providers have different (proprietary) interpretations of various error codes
  - Difficult to find out why ibv_post_send() or ibv_poll_cq() failed, for example
- Perhaps a better strerr() type of functionality (that can also obtain provider-specific strings)?

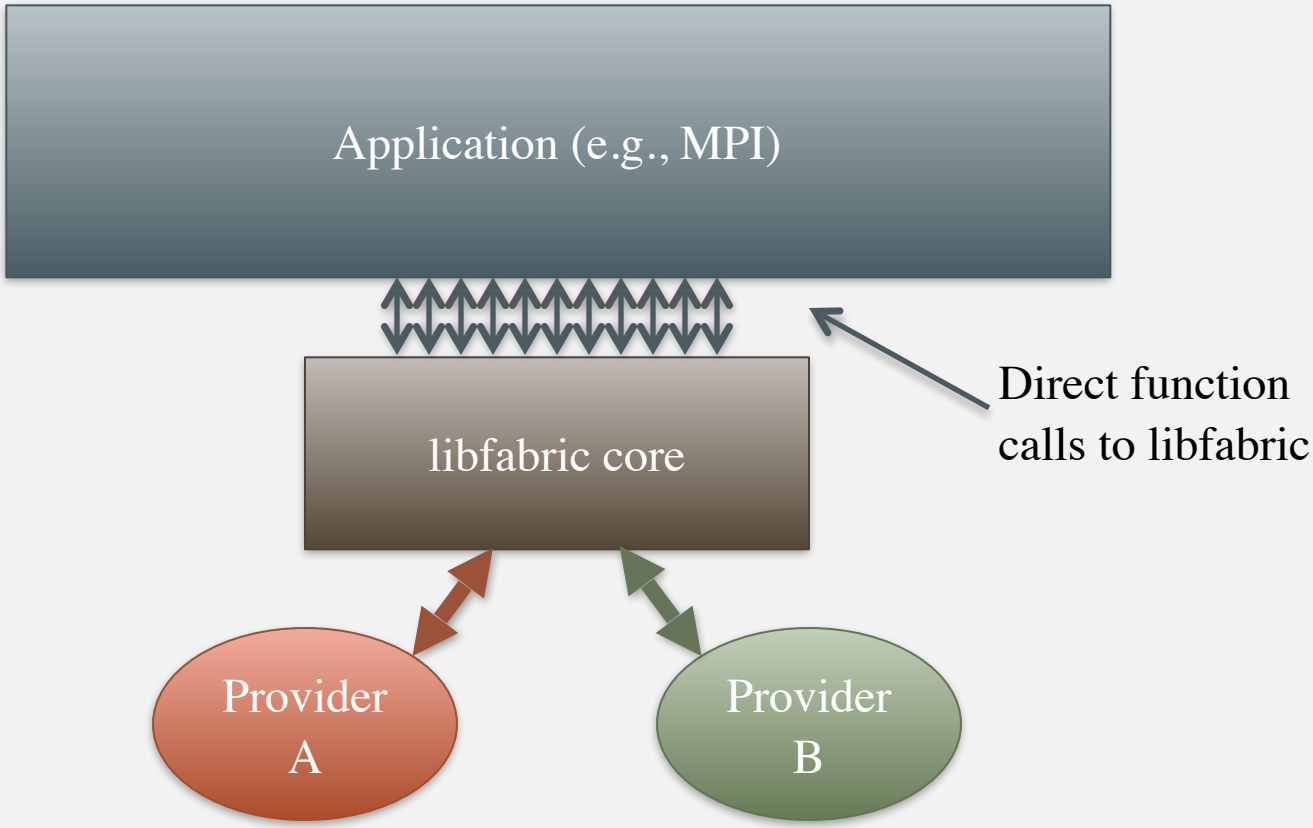
Other things MPI wants: Standardized high-level interfaces

- Examples:
  - Tag matching
  - MPI non-blocking collective operations (TBD)
  - Remote atomic operations
  - …etc.
  - *The MPI community wants input in the design of these interfaces*
- Divided opinions from MPI community:
  - Providers must support these interfaces, even if emulated
  - Run-time query to see which interfaces are supported

Other things MPI wants: Vendor-specific interfaces

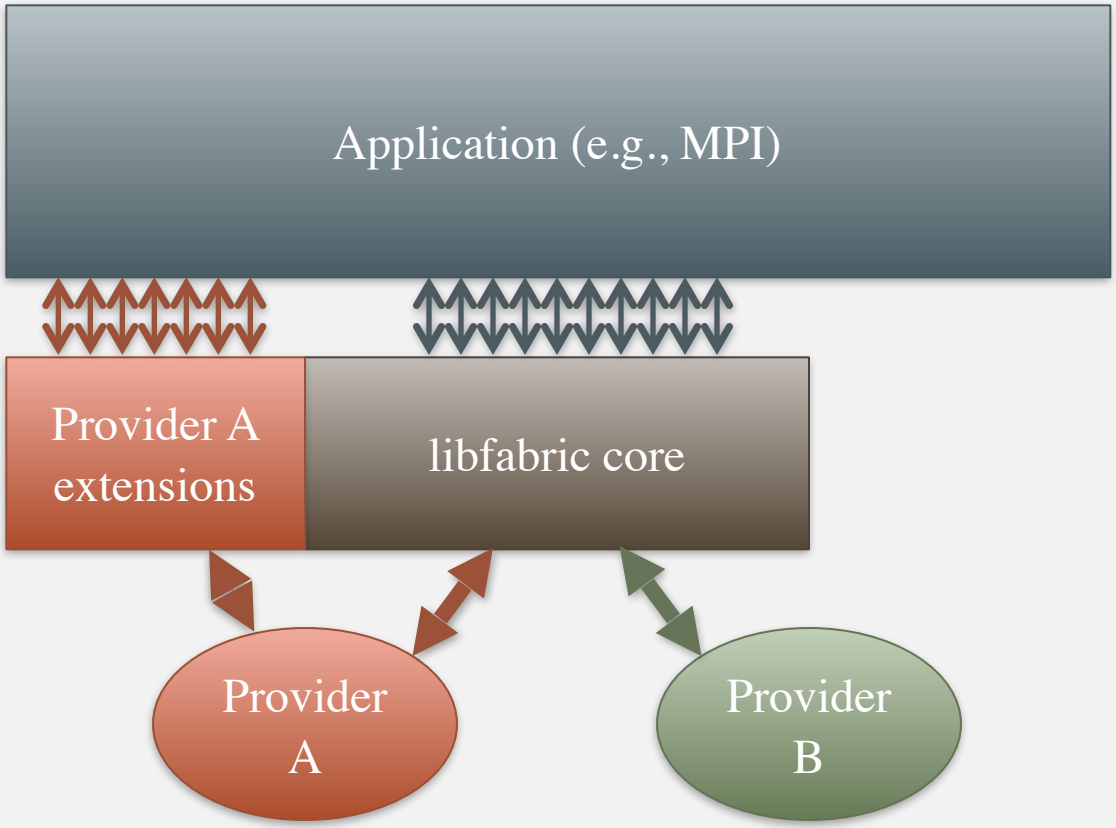
- Direct access to vendor-specific features
  - *Lowest-common denominator API is not always enough*
  - Allow all providers to *extend* all parts of the API
- Implies:
  - Robust API to query what devices and providers are available at run-time (and their various versions, etc.)
  - Compile-time conventions and protections to allow for safe non-portable codes
- *This is a radical difference from verbs*

Core libfabric functionality

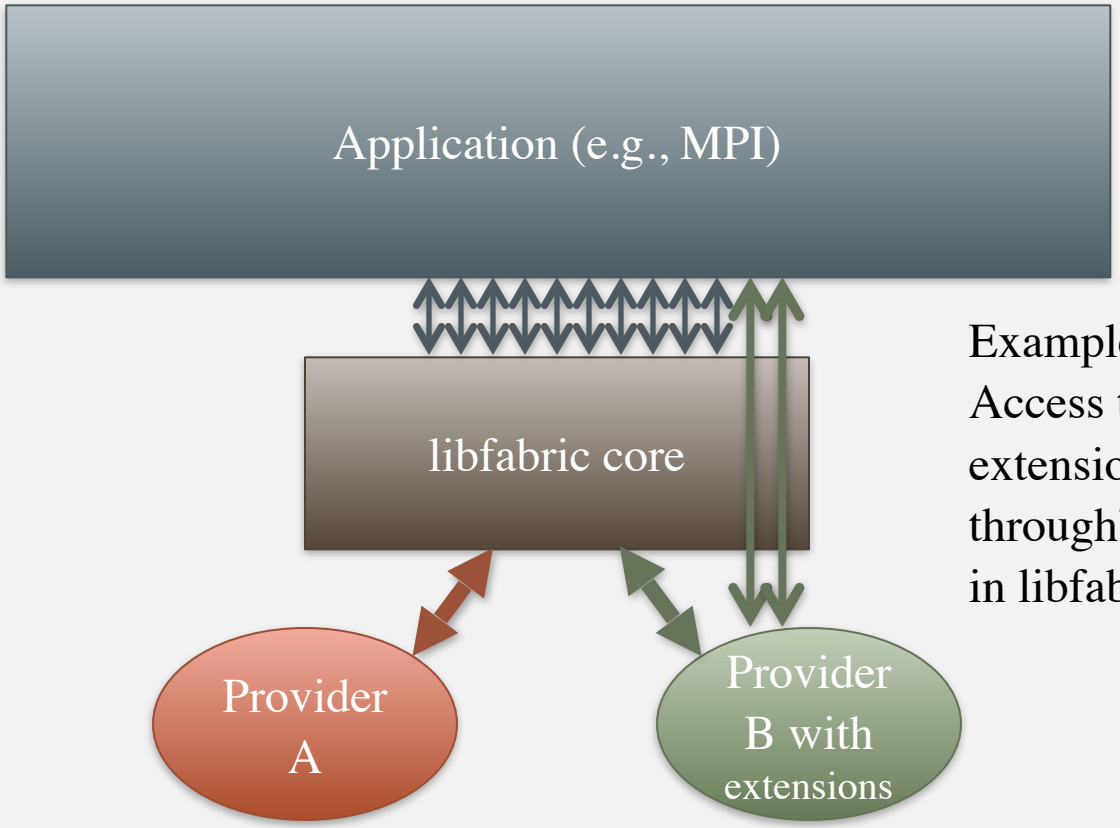


Example options for direct access to vendor-specific functionality

Example 1: Access to provider A extensions without going through libfabric core



Example options for direct access to vendor-specific functionality



Example 2:
Access to provider B extensions via "pass through" functionality in libfabric

Other things MPI wants: Regarding memory registration

- Run-time query: is memory registration is necessary?
  - I.e., explicit or implicit memory registration
- If explicit
  - Need robust notification of involuntary memory de-registration (e.g., munmap)
- If the cost of de/registration were “free”, much of this debate would go away ☺

Other things MPI wants: Regarding fork() behavior

- In child:
  - All memory is accessible (no side effects)
  - Network handles are stale / unusable
  - Can re-initialize network API (i.e., get new handles)
- In parent:
  - All memory is accessible
  - Network layer is still fully usable
  - Independent of child process effects

Other things MPI wants

- If network header knowledge is required:
  - Provide a run-time query
  - Do not mandate a specific network header
  - E.g., incoming verbs datagrams require a GRH header
- Request ordered vs. unordered delivery
  - Potentially by traffic type (e.g., send/receive vs. RDMA)
- Completions on both sides of a remote write

Other things MPI wants

- Allow listeners to request a specific network address
  - Similar to TCP sockets asking for a specific port
- Allow receiver providers to consume buffering directly related to the size of incoming messages
  - Example: “slab” buffering schemes

Other things MPI wants

- Generic completion types. Example:
  - Aggregate completions
  - Vendor-specific events
- Out-of-band messaging

Other things MPI wants

- Noncontiguous sends, receives, and RDMA opns.
- Page size irrelevance
  - Send / receive from memory, regardless of page size
- Access to underlying performance counters
  - For MPI implementers and MPI-3 “MPI_T” tools
- Set / get network quality of service

Other things MPI wants: More atomic operations

- Datatypes (minimum): int64_t, uint64_t, int32_t, uint32_t
  - Would be *great*: all C types (to include double complex)
  - Would be *ok*: all <stdint.h> types
  - Don't require more than natural C alignment
- Operations (minimum)
  - accumulate, fetch-and-accumulate, swap, compare-and-swap
- Accumulate operators (minimum)
  - add, subtract, or, xor, and, min, max
- Run-time query: are these atomics coherent with the host?
  - If support both, have ability to request one or the other

Other things MPI wants: MPI RMA requirements

- Offset-based communication (not address-based)
  - Performance improvement: potentially reduces cache misses associated with offset-to-address lookup
- Programmatic support to discover if VA based RMA performs worse/better than offset based
  - Both models could be available in the API
  - But not required to be supported simultaneously
- Aggregate completions for MPI Put/Get operations
  - Per endpoint
  - Per memory region

Other things MPI wants: MPI RMA requirements

- Ability to specify remote keys when registering
  - Improves MPI collective memory window allocation scalability
- Ability to specify arbitrary-sized atomic ops
  - Run-time query supported size
- Ability to specify/query ordering and ordering limits of atomics
  - Ordering mode: rar, raw, war and waw
  - Example: “rar” – reads after reads are ordered

“New,” but becoming important

- Network topology discovery and awareness
  - …but this is (somewhat) a New Thing
  - Not much commonality across MPI implementations
- Would be nice to see some aspect of libfabric provide fabric topology and other/meta information
  - Need read-only access for regular users

API design considerations

- With no tag matching, MPI frequently sends / receives two buffers
  - (header + payload)
  - Optimize for that
- MPI sometimes needs thread safety, sometimes not
  - May need both in a single process
- Support for checkpoint/restart is desirable
  - Make it safe to close stale handles, reclaim resources

API design considerations

- Do not assume:
  - Max size of any transfer (e.g., inline)
  - The memory translation unit is in network hardware
  - All communication buffers are in main RAM
  - Onload / offload, but allow for both
  - API handles refer to unique hardware resources
- Be “as reliable as sockets” (e.g., if a peer disappears)
  - Have well-defined failure semantics
  - Have ability to reclaim resources on failure

Conclusions

- Many different requirements
  - High-level, low-level, and vendor-specific interfaces
- The MPI community would like to continue to collaborate
  - Tag matching is well-understood, but agreeing on a common set of interfaces for them will take work
  - Creating other high-level MPI-friendly interfaces (e.g., for collectives) will take additional work

Thank you!